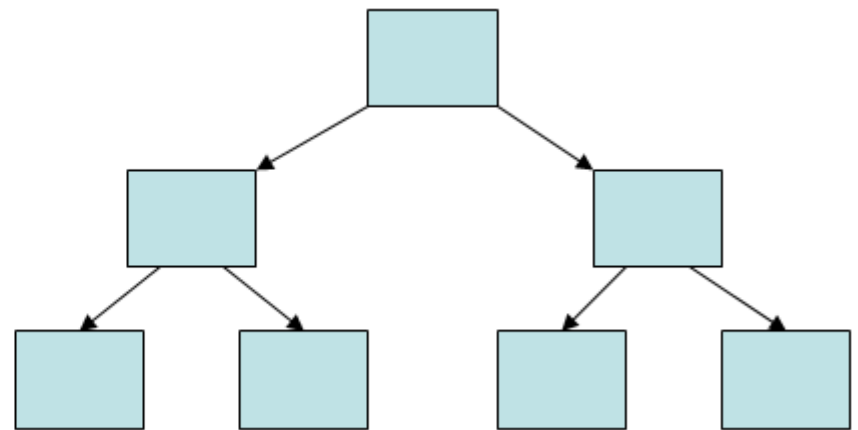
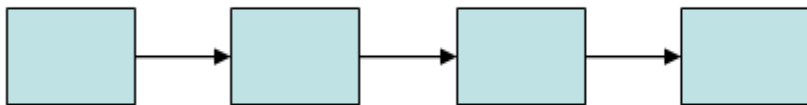


Dynamische Datenstrukturen



Beispiel: Adressbuch

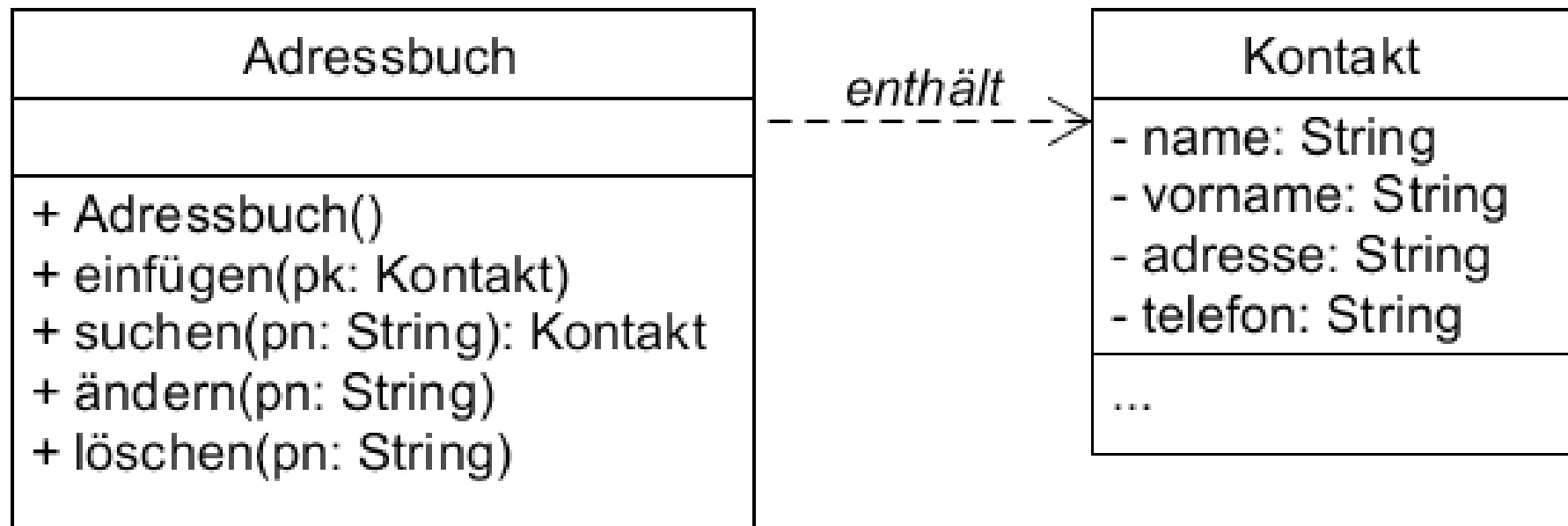
Speichert Kontakte, z.B. einer Firma.

Es kommen laufend neue Adressen hinzu.

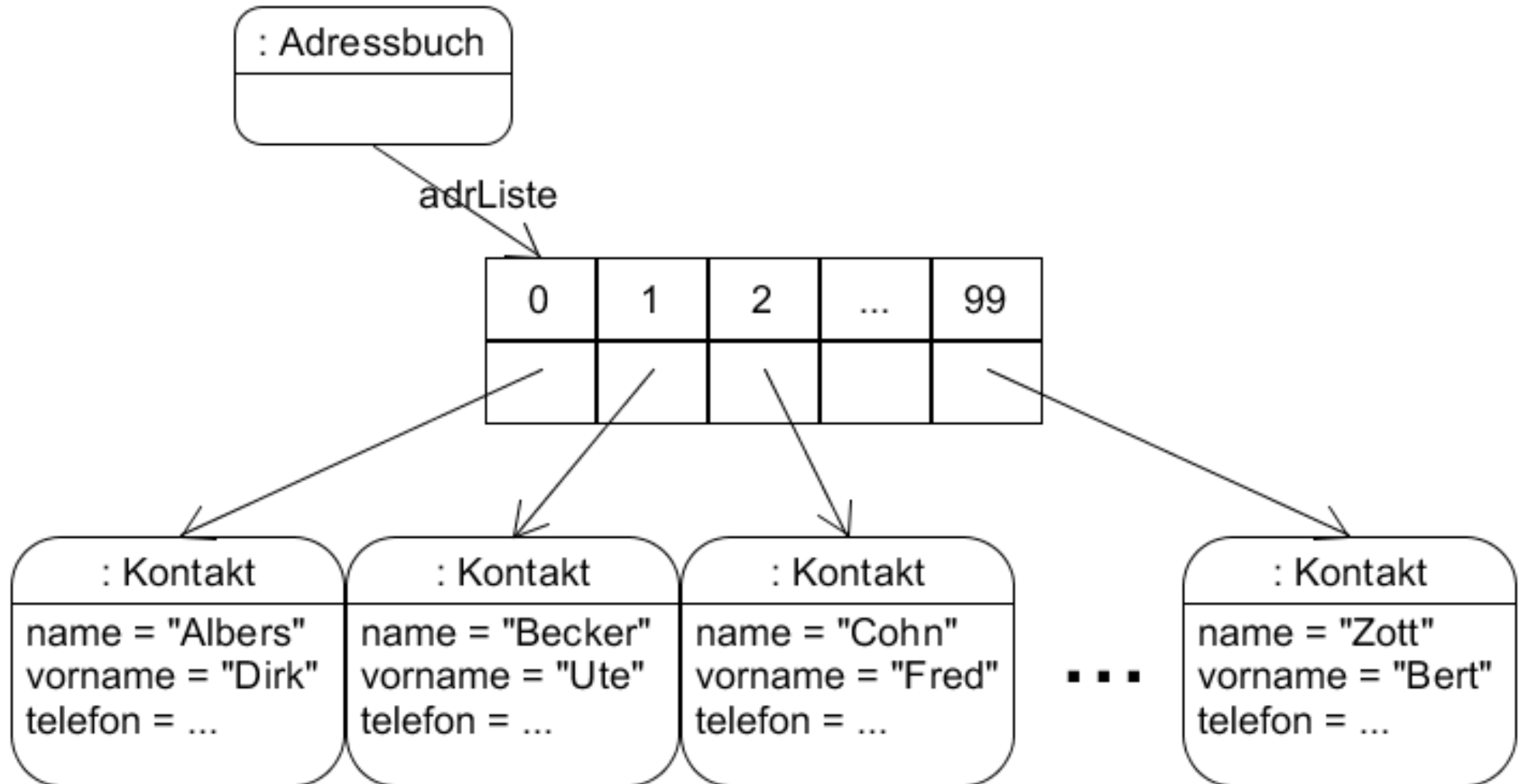
Namen und Adressen können sich ändern.

Nicht mehr benötigte Adressen werden gelöscht.

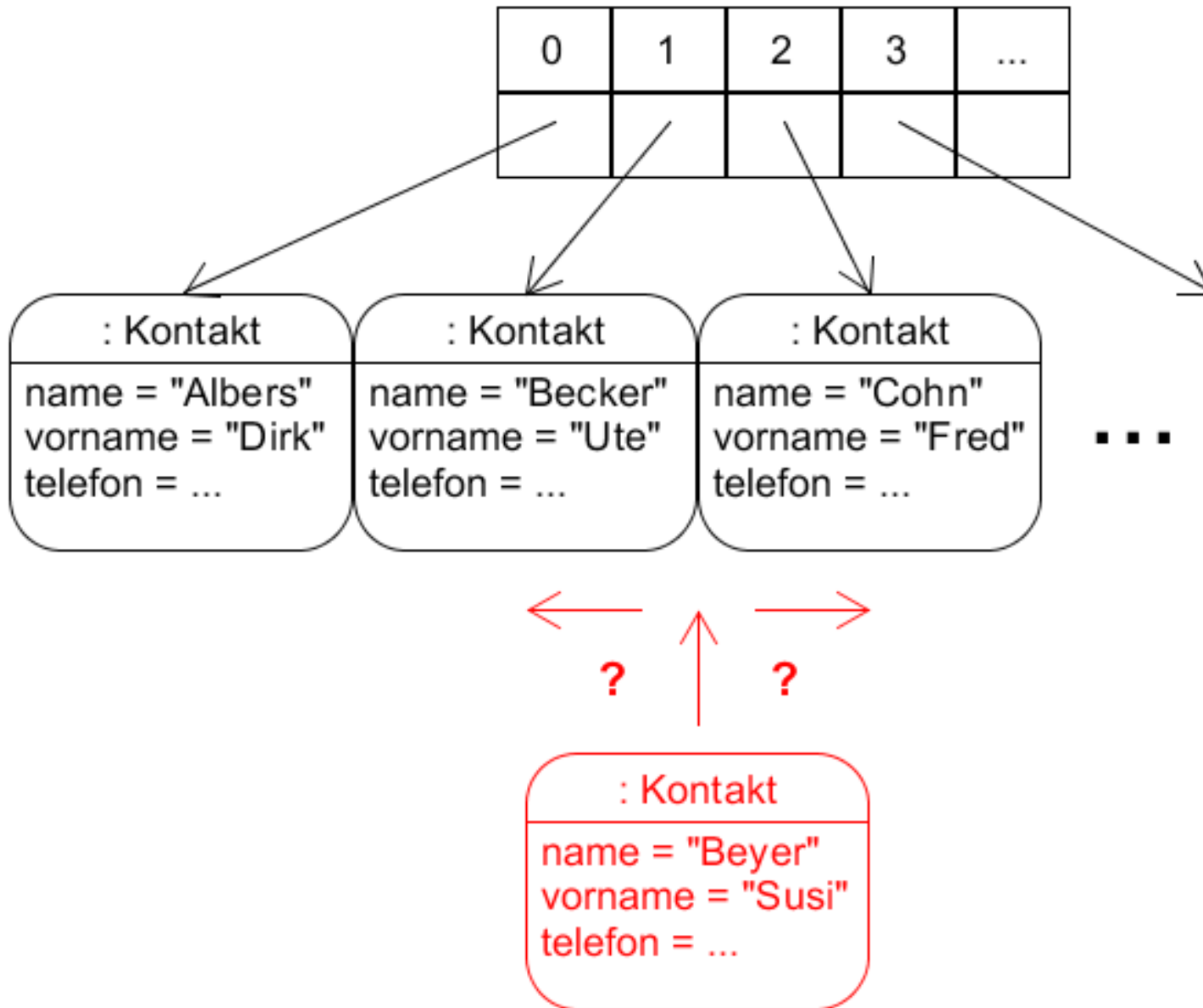
Das Adressbuch soll alphabetisch sortiert sein.



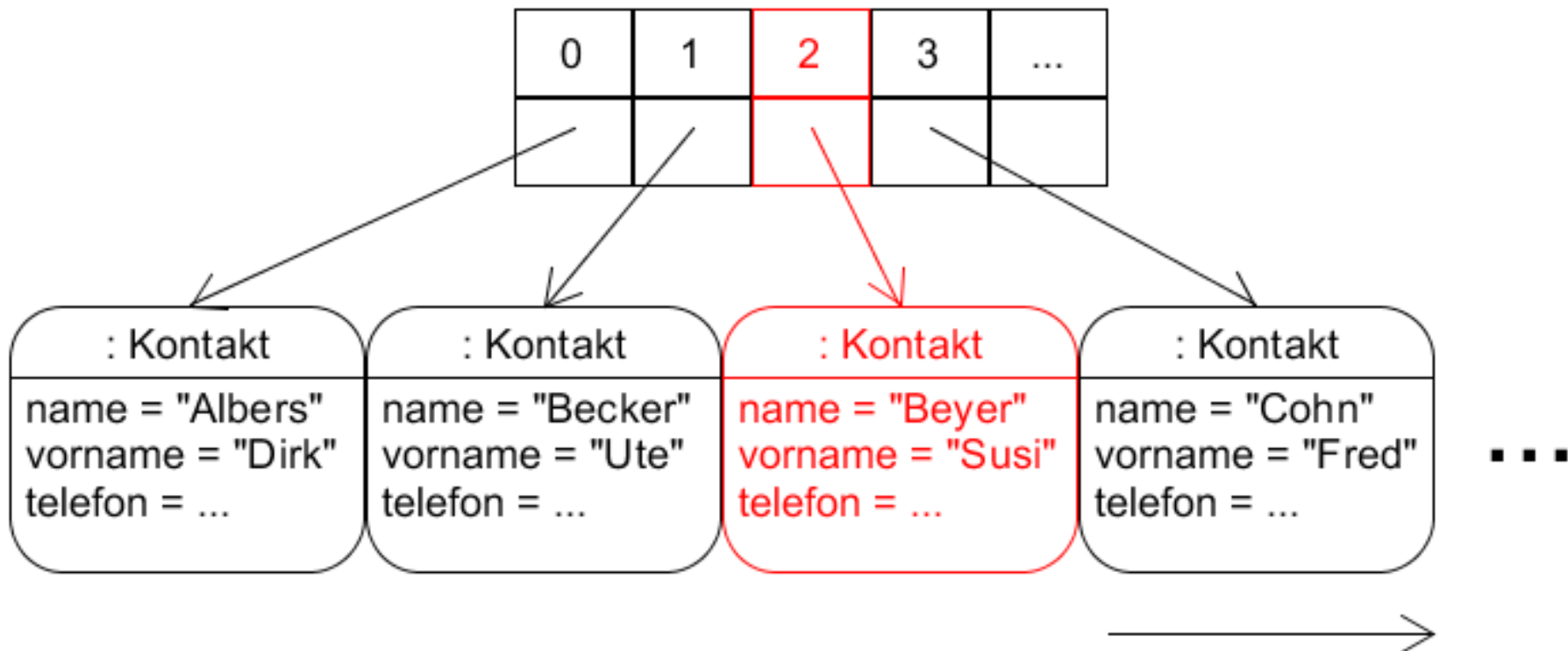
Implementierung als Array



Problem 1: Einfügen



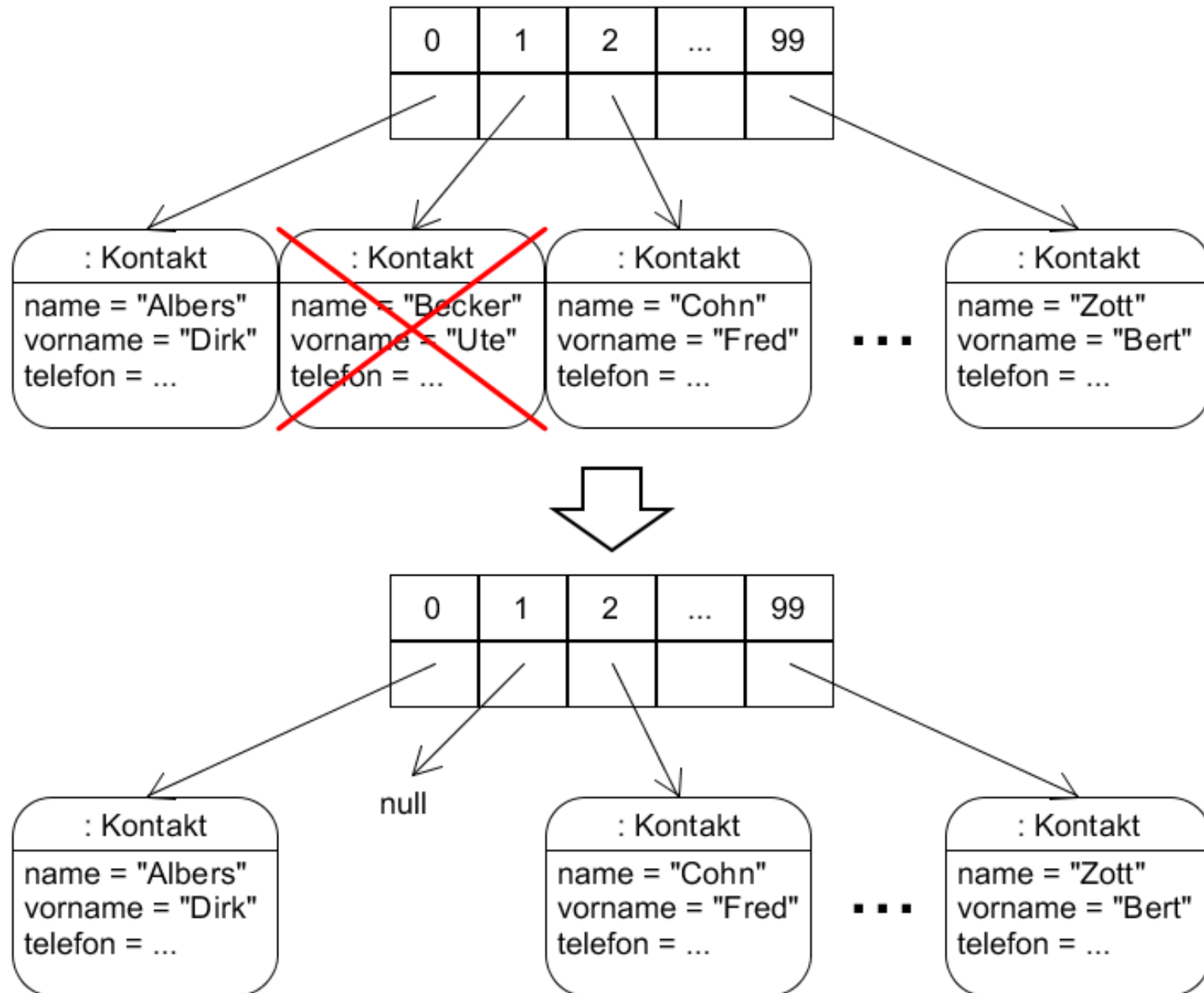
Problem 1: Einfügen



Lösung: Richtige Stelle finden
alle Nachfolger eine Position nach rechts schieben

Nachteil: aufwendig

Problem 2: Löschen



Problem 2: Löschen

Lösung 1: Lücke lassen
(um später neue Adresse einzufügen)

Nachteil: mehr und mehr Lücken

Lösung 2: Lücke schließen
(Nachfolger eine Position nach links rücken)

Nachteil: aufwendig

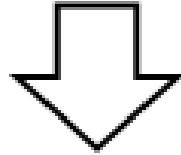
Problem 3: Liste voll

Die „Länge“ eines Arrays ist begrenzt.

Wenn es voll ist, können keine weiteren Kontakte eingefügt werden.

Problem 3: Liste voll

0	1	2	...	99



0	1	2	...	99	100	101	102	...	199

Lösung: größeres Array erzeugen, altes kopieren

Nachteile: aufwendig, kostet Speicherplatz

Dynamische Datenstrukturen

- Größe passt sich nach Bedarf flexibel an
- Elemente einfügen / löschen an beliebiger Stelle (ohne Verschieben anderer Elemente)

→ haben nicht die Probleme von Arrays

Varianten:

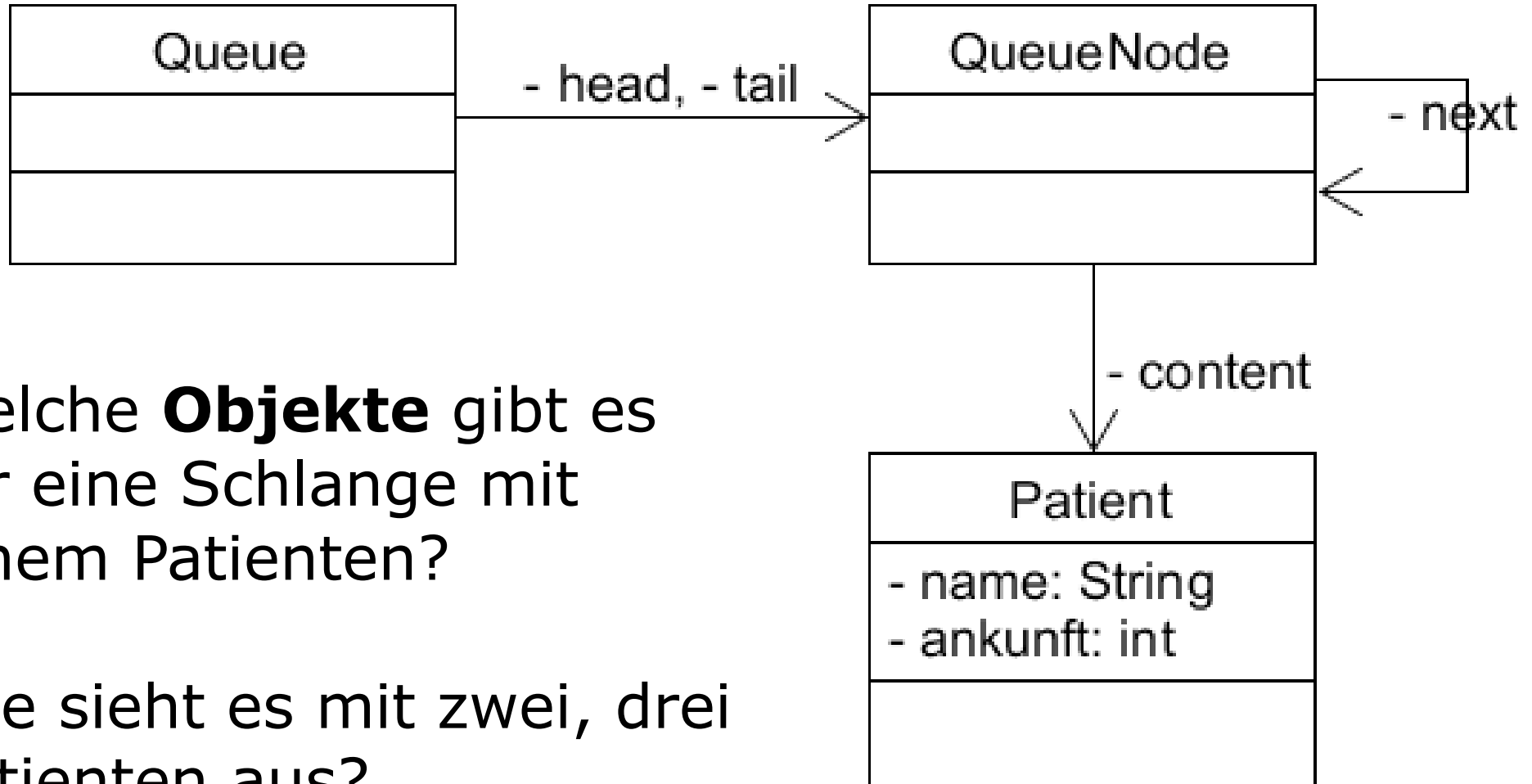
- Liste
- Warteschlange (Queue)
- Stapel (Stack)
- (Binär-)baum
- Netzwerk (Graph)

Beispiel: Warteschlange beim Arzt



Morgens: Schlange ist leer, bis 1. Patient kommt
Jeder neue Patient stellt sich ans Ende der Schlange.
Wer am längsten gewartet hat, ist an der Reihe.

Datenstruktur „Queue“



Welche **Objekte** gibt es für eine Schlange mit einem Patienten?

Wie sieht es mit zwei, drei Patienten aus?

queue = Schlange node = Knoten content = Inhalt
head = Kopf (Anfang) tail = Schwanz (Ende)
ankunft: fortlaufende Nummer

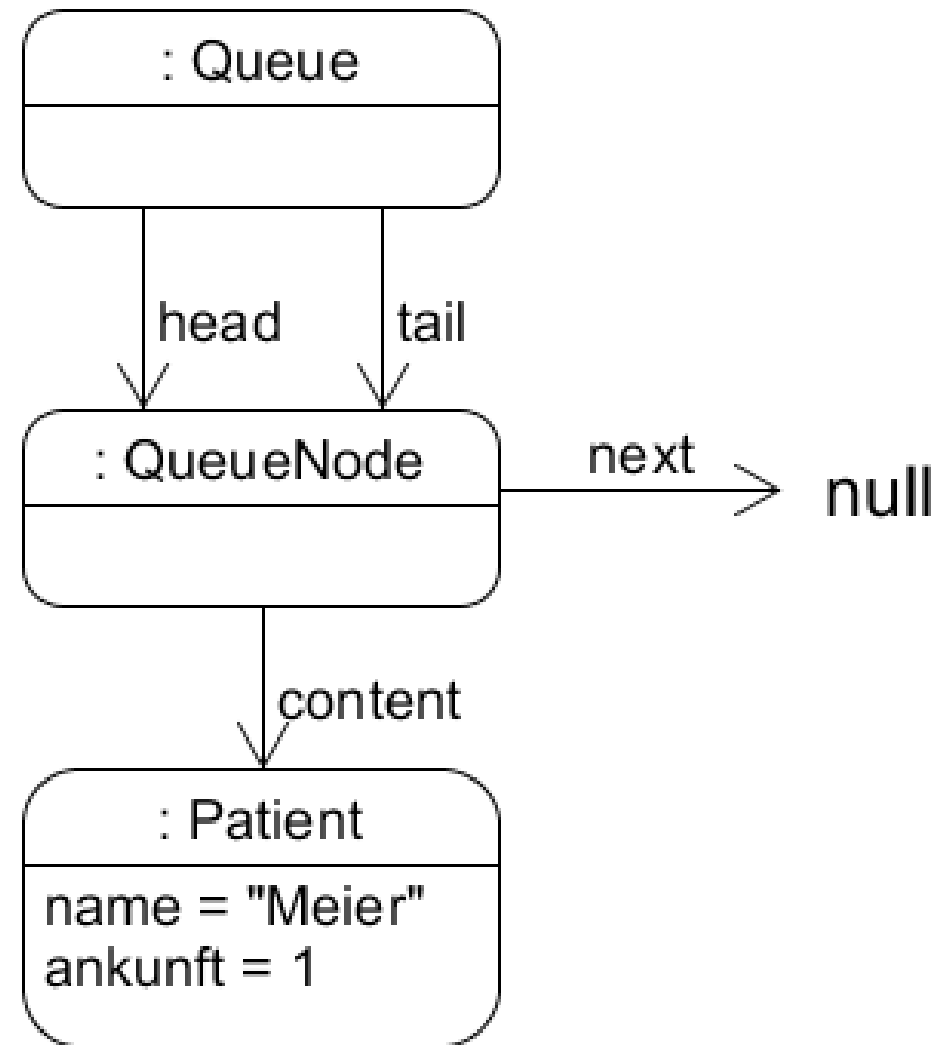
Objektdiagramm

QueueNode-Objekt
hat Referenz auf
Patient-Objekt

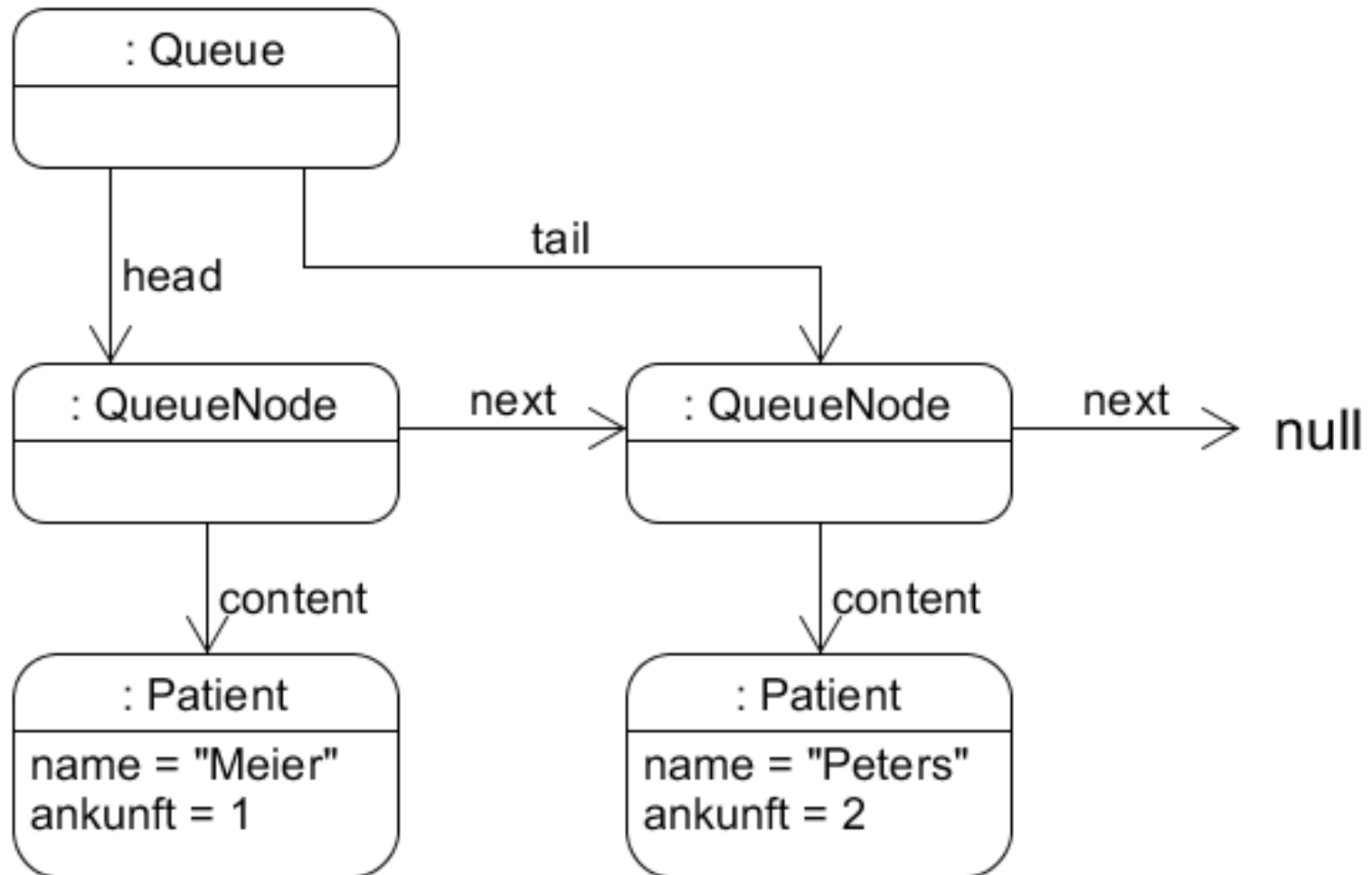
Queue-Objekt
hat zwei Referenzen auf
QueueNode-Objekte:
head und tail

Ein Patient
→ nur ein Node
→ head **und** tail zeigen
auf diesen Node

Keine weiteren Nodes: next = null



Objektdiagramm

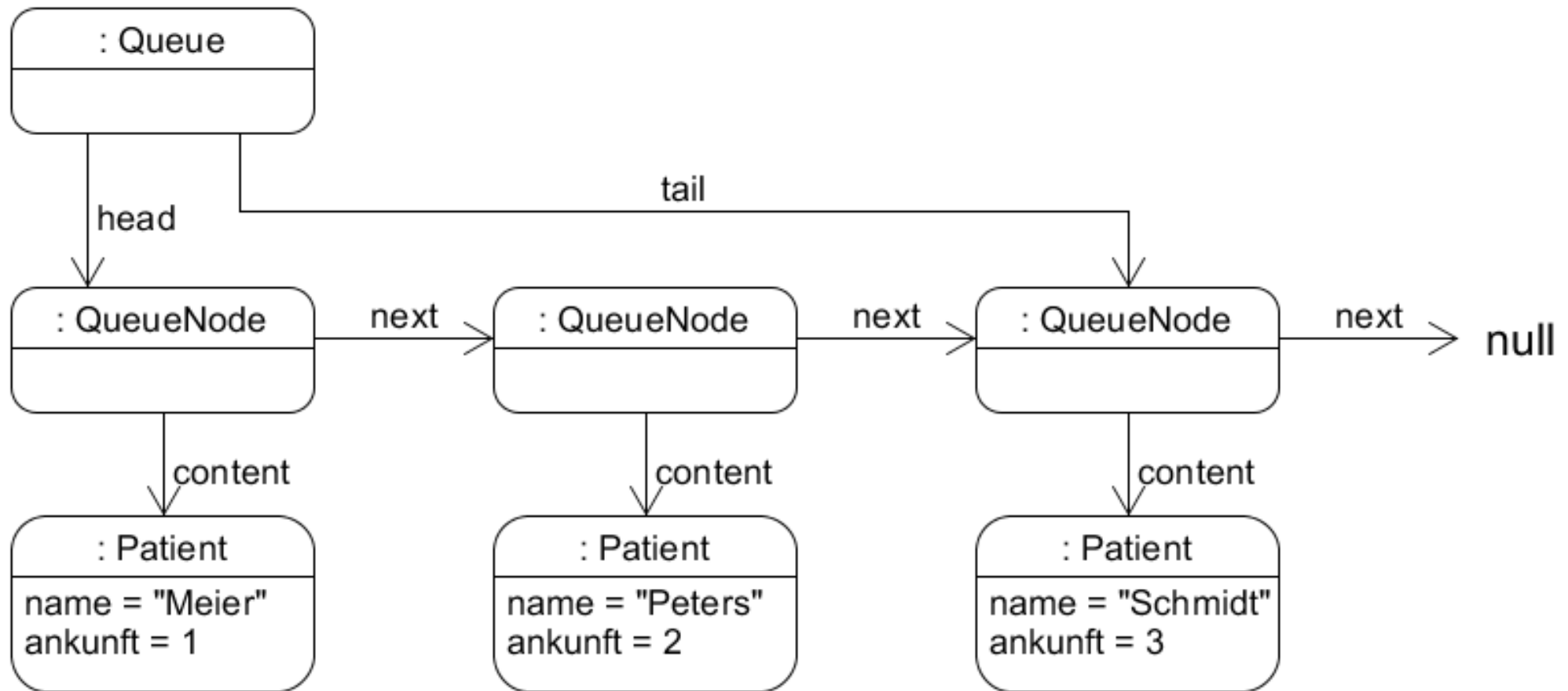


Nächster Patient:

Neues QueueNode-Objekt angehängt

tail verschiebt sich auf dieses QueueNode-Objekt.

Objektdiagramm

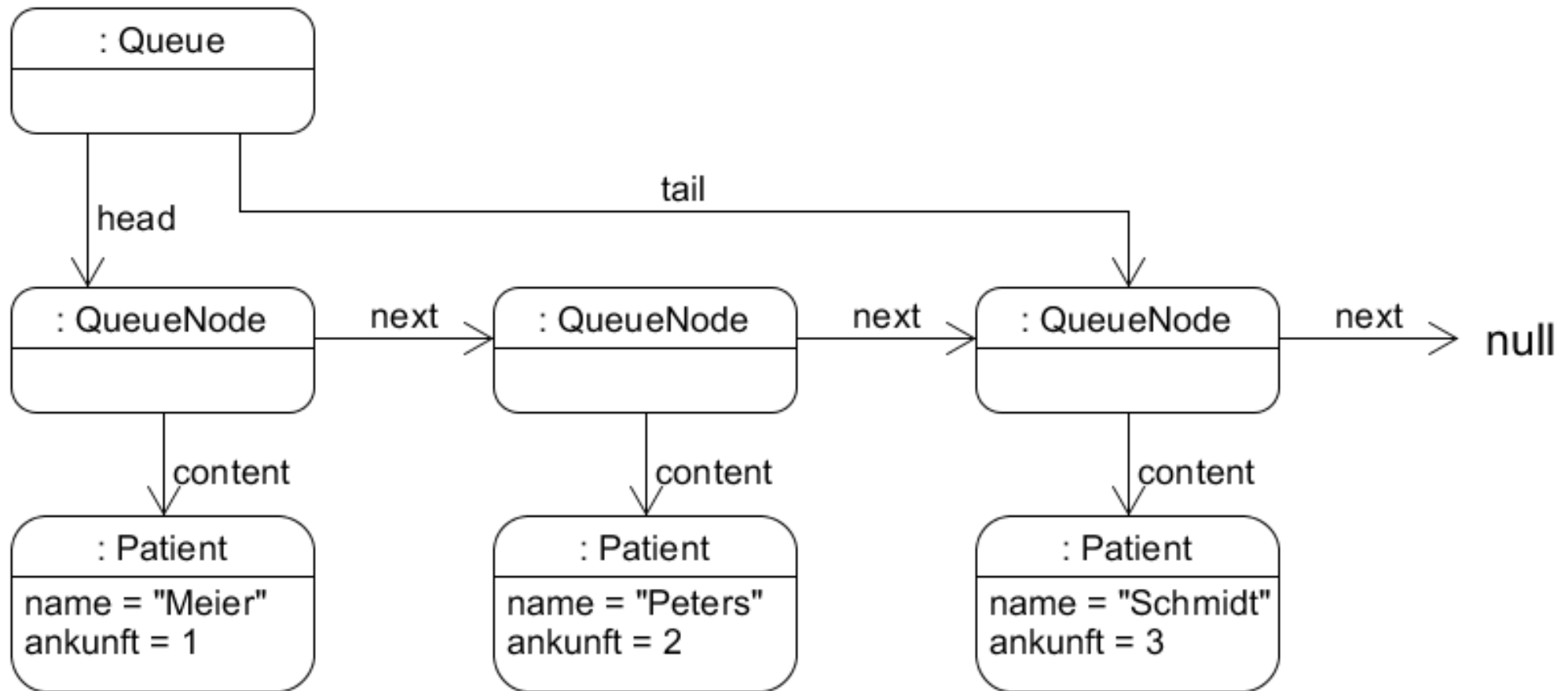


Nächster Patient:

Neues QueueNode-Objekt angehängt

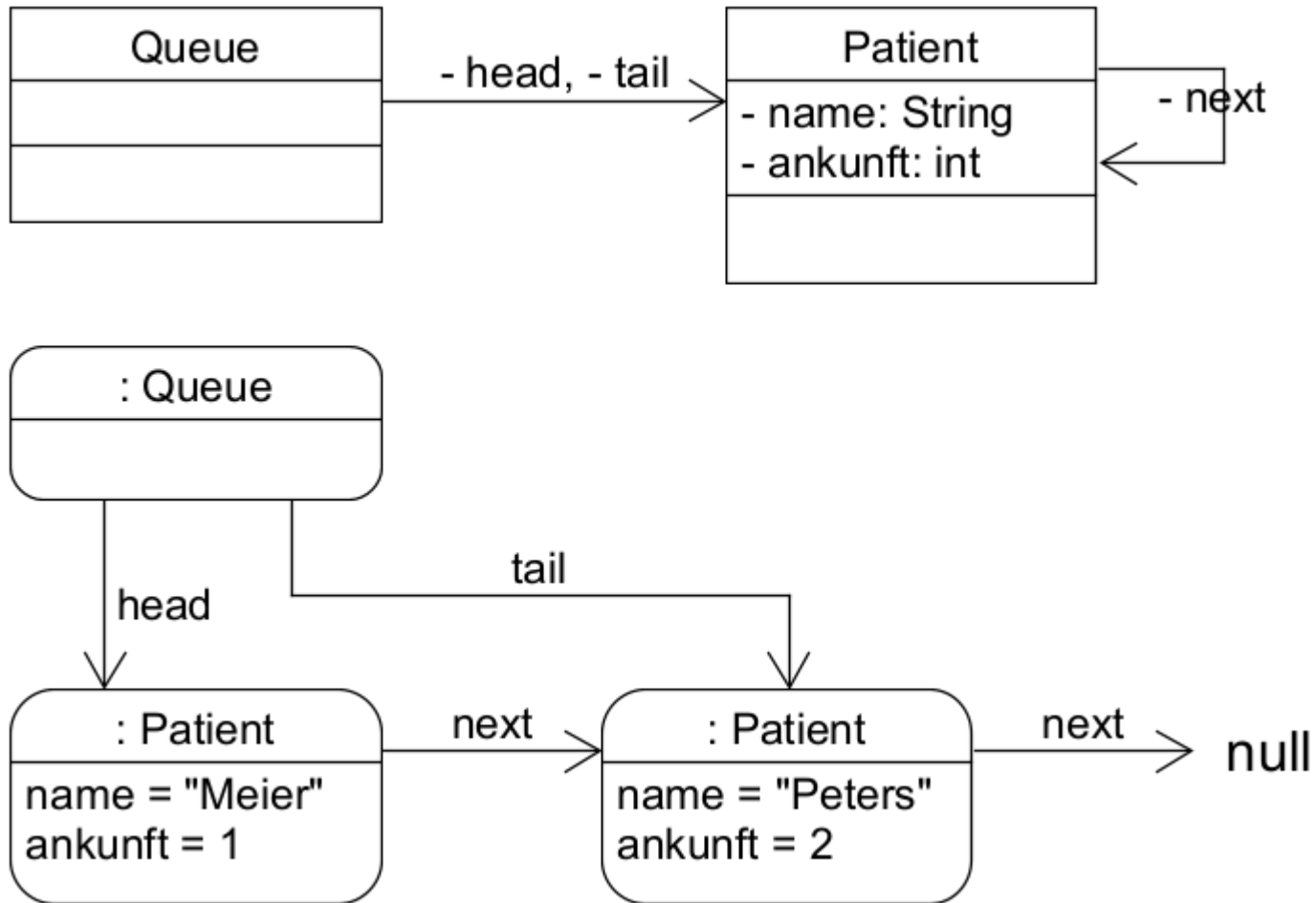
tail verschiebt sich auf dieses QueueNode-Objekt.

Objektdiagramm

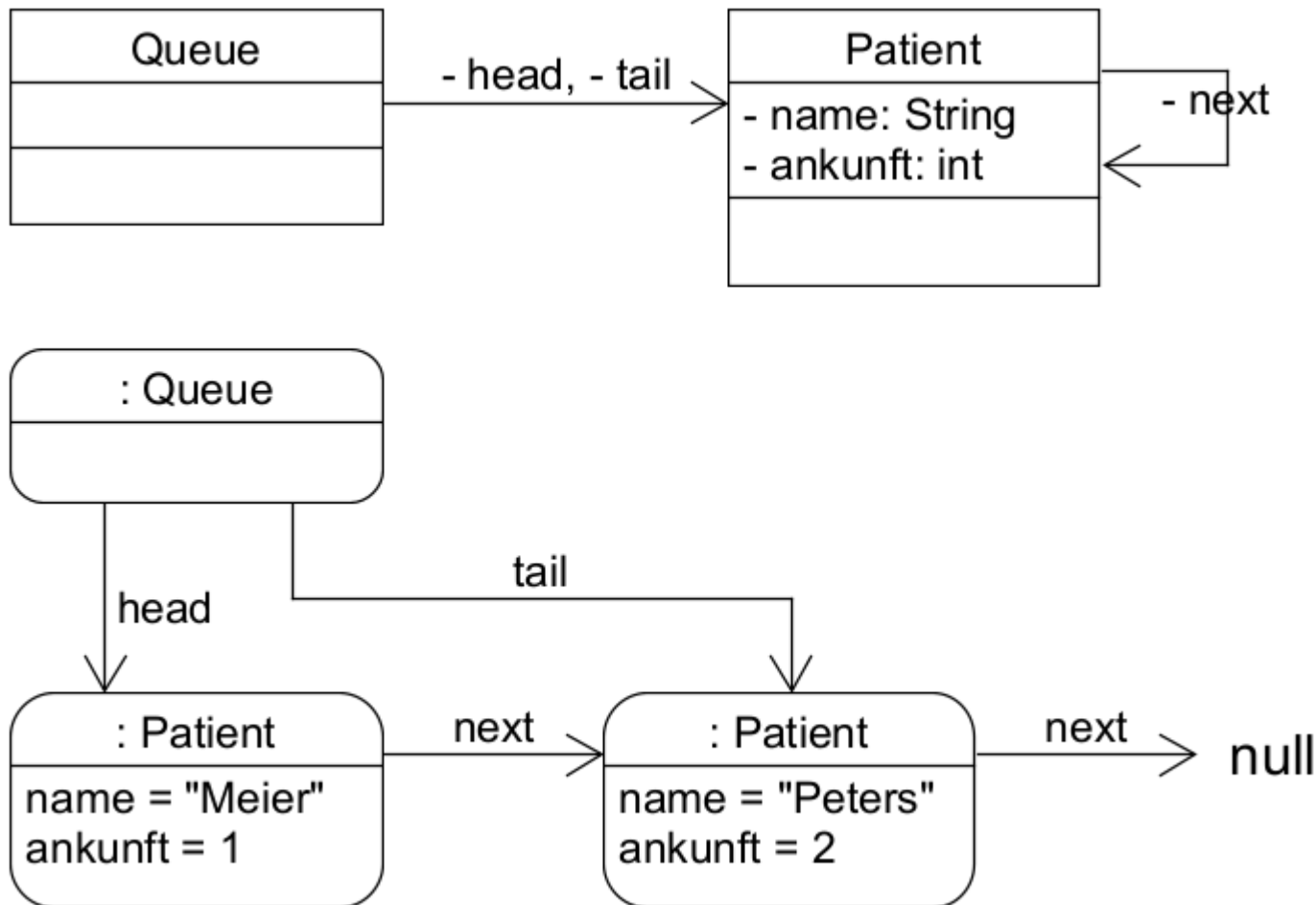


Aufgabe: Der erste Patient verlässt die Schlange.
Wie ändert sich das Diagramm?
Wie sieht es aus, wenn alle drei behandelt wurden?

Warum nicht einfach so?



Warum nicht einfach so?



So müsste man die Queue-Funktionen für jede Anwendung neu programmieren.
OOP bedeutet klare Aufteilung nach Funktion.

Anwendungsbeispiele

Die Datenstruktur Queue ist nützlich, wenn Dinge in der gleichen Reihenfolge bearbeitet werden, in der sie „eingegangen“ sind.

Beispiele

- Anrufe im Callcenter
- Anfragen an einen Webserver
- Steuerung eines Online-Spiels

Die Queue entspricht dem **FIFO-Prinzip**:

- First in, first out

Autor / Quellen

Autor:

- Christian Pothmann (cpothmann.de)
Freigegeben unter CC BY-NC-SA 4.0, Mai 2021



Grafik:

- Zeichnung Warteschlange: Wellcome Images, London
Freigegeben unter CC BY 4.0